

GPIO Module Documentation

1. GPIO Architecture Document

1.1 Purpose

The GPIO (General Purpose Input/Output) module provides a deterministic, statically configured abstraction for controlling microcontroller digital I/O pins. It is designed for bare-metal embedded systems where predictability, safety, and register-level control are required.

The module allows configuration and runtime control of GPIO pins while isolating hardware-specific register details from higher software layers.

1.2 Scope

Included in scope: - GPIO pin initialization - Pin direction, mode, speed, pull configuration - Alternate function configuration - Runtime pin read/write APIs - Module state management

Explicitly out of scope: - Clock enabling and clock tree configuration - Interrupt (EXTI) configuration - Power management and low-power modes - OS-specific services

1.3 Architectural Style

The GPIO module follows a **Layered, Table-Driven Driver Architecture** with compile-time configuration.

Layers: - Application Layer - GPIO Public API Layer (`GPIO.h`) - GPIO Internal Services (`GPIO_Int.h`) - Hardware Register Access Layer (`Registers_Table`)

Configuration is externalized as static constant data and consumed during initialization.

1.4 Configuration Model

- GPIO configuration is **table-driven** using compile-time constant structures.
- Each GPIO pin is described by one configuration entry.
- No runtime modification of configuration data is allowed.
- Configuration includes:
 - Port ID
 - Pin ID
 - Mode
 - Output type

- Speed
- Pull-up/pull-down
- Alternate function
- Default output state

This model enables deterministic initialization and eliminates duplicated configuration code.

1.5 Initialization Model

- The GPIO module exposes a single initialization API (`GPIO_Init`).
 - Initialization iterates over the configuration table and programs hardware registers accordingly.
 - The module assumes that the corresponding GPIO clocks are enabled before initialization.
 - After successful initialization, the module transitions to the **INITIALIZED/READY** state.
-

1.6 Module State Management

The GPIO module maintains an internal state machine:

- RESET: Module not initialized
- INITIALIZED: Configuration applied
- READY: Module available for runtime use
- ERROR: Invalid operation detected

All public APIs validate the current module state before accessing hardware registers.

1.7 Register Access Strategy

- Atomic operations (BSRR) are used for setting and resetting output pins.
 - Read-Modify-Write (RMW) access is used only where required (e.g., MODER, AFR).
 - Direct writes to ODR for output control are avoided.
 - Register access is abstracted through a register mapping table to centralize hardware knowledge.
-

1.8 Error Handling Strategy

- All public APIs perform parameter validation.
 - Invalid pin IDs or invalid module state result in defined error return values.
 - No silent failures or undefined behavior are permitted.
-

1.9 Extensibility

The architecture supports:

- Adding new ports or pins via configuration only
- Integration with RTOS environments
- Future extensions such as power management or safety checks

No architectural changes are required to scale the module.

2. GPIO Requirements Document

2.1 Functional Requirements

- GPIO shall support configuration of digital pins during system initialization.
 - GPIO shall support runtime reading of pin logical state.
 - GPIO shall support runtime writing of pin logical state.
 - GPIO shall support configuration of alternate functions for peripheral usage.
 - GPIO shall support configuration of pull-up and pull-down resistors.
-

2.2 Non-Functional Requirements

- GPIO operations shall be deterministic in execution time.
 - GPIO shall not use dynamic memory allocation.
 - GPIO shall be fully configurable at compile time.
 - GPIO APIs shall be safe for use in interrupt context where applicable.
 - GPIO write operations shall be atomic where hardware supports it.
-

2.3 Initialization Requirements

- GPIO shall not allow runtime API usage before initialization.
 - GPIO initialization shall assume GPIO clocks are enabled beforehand.
 - GPIO shall transition to a valid operational state after successful initialization.
-

2.4 State Management Requirements

- GPIO shall internally track its initialization state.
 - GPIO shall reject API calls made in invalid states.
 - GPIO shall expose status information for diagnostic purposes if required.
-

2.5 Error Handling Requirements

- GPIO shall validate all input parameters.
 - GPIO shall return explicit error codes on failure.
 - GPIO shall not cause undefined hardware behavior on invalid usage.
-

2.6 Constraints

- Target platform: STM32-class microcontrollers
 - Programming language: C
 - No operating system dependency
 - Register-level hardware access required
-

2.7 Compliance and Testability

- All requirements shall be traceable to implementation.
 - Configuration-driven design enables static analysis and unit testing.
 - Module behavior shall be verifiable through register inspection and hardware tests.
-

End of Document